Technical Report 183/1-99, Dept. of Automation, TU Vienna, February 2000

Security in Distributed Systems - A Survey

Bettina Weiss

October 9, 2002

Contents

Abstract

This report surveys the existing literature on security issues in distributed systems. It aims to give an overview on existing protocols and current research.

1 Introduction

In our project W2F[1] (Wireless/Wired Factory/Facility Fieldbus), we intend to develop a next-generation LAN/fieldbus which will properly address distribution, security, fault-tolerance, and real-time issues as well as flexibility w.r.t. wireline/wireless interconnections. In the security field, we will have several decisions to make:

- What requirements to security do we have?
- Which security protocols should we implement?
- Should we use existing protocols or develop new ones?
- How can we prove that the protocols we choose are fault-free?

The following report surveys existing security methods relevant to these issues. In Section 2, we explain what requirements to security protocols exist, and in Section 3 we examine current design methods. We look at hardware solutions to security issues in Section 4 and discuss the basic structure of software protocols in Section 5. In the concluding Section 6 we briefly examine the appropriateness of the surveyed methods for W2F. Appendix A lists some popular encryption schemes, and Appendix B presents some existing security protocols.

Before we adress these issues, let us briefly review why we need security in the first place. We assume a distributed computing environment, with n nodes which can only communicate over a network. In order to provide or use a distributed service the nodes need to exchange information. However, sending a message in an arbitrary environment rises several potential problems:

- Any node in the communication path from the sender to the receiver can read the message.
- In a partially connected network, the message can be intercepted and removed by an adversary.
- In a partially connected network, the message can be replaced by an adversary. The adversary can alter the sender, the receiver, and/or the contents of the message.

[1] This work is part of our W2F-project (*http://www.auto.tuwien.ac.at/Projects/W2F/*), which received funding from the Austrian START-programme Y41-MAT.

- An adversary can generate forged messages.

Security protocols aim to guarantee to the sender and the receiver that some or all of the above-mentioned problems either cannot occur or are at least noticed. For our own project, we will have to identify which of these requirements we have to meet and how we should meet them.

2 Protocol Goals

Although the term "security protocol" is commonly used for a wide range of protocols, the goals of these protocols can differ significantly. Different situations require different demands on the security policy and not every protocol needs to implement every possible security mechanism to achieve its goals. We believe that it is important to discern between the requirements of the security policy and the goals of a particular protocol. The former are global issues, which can be enforced by using one or more security protocols. The latter are the goals of one protocol and may differ between different protocols. The following list explains some possible requirements of a security policy.

Accounting logs service requests of users. The goals are manifold, ranging from the ability to trace back system attacks up to the maintainance of disk quotas for each client. The service depends on authentication.

Authentication aims to guarantee the identity of principals during communication. In *one-way* authentication, only one of the partners is sure about the identity of the other, in *two-way* authentication both peers know with whom they are talking.

Authorization strives to grant access only to special groups. The requirement depends on authentication to ensure who the principal is, but additionally has to check whether the principal is allowed to access given services.

Integrity guarantees that message modifications are either impossible or can at least be detected by the receiver.

Nonrepudiation prohibits a sender from later denying that it has sent a message.

Secrecy tries to keep the contents of a communication exchange hidden from possible spies. There are different degrees of secrecy, from simple encryption of the message's contents up to the hiding of message senders and recipients.

In contrast, protocol goals are concerned with particular problems that arise when implementing the above requirements (e.g., secrecy is achieved by an encryption protocol which in turn requires a key exchange protocol). Boyd [Boy97] has examined the goals which are normally associated with key distribution and authentication protocols. He follows the classification of Roscoe [Ros96] who has parted goals into *extensional* and *intensional* ones. Extensional goals are abstract and therefore independent of the protocol, whereas intensional goals are protocol specific and concerned with the way particular protocol states are reached. Boyd lists possible extensional goals for key distribution and authentication and orders them into a hierarchy, see Figure 1.

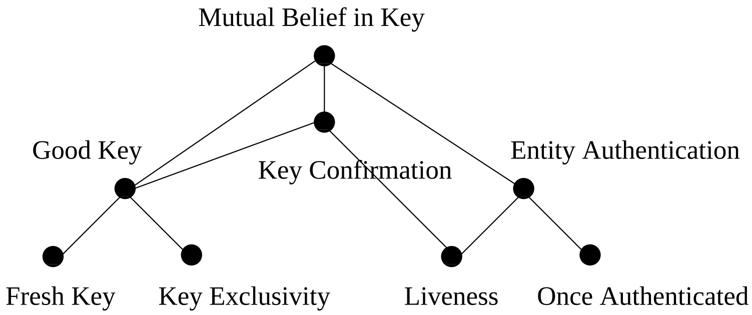

Figure 1: *Hierarchy of extensional goals*

Good Key: A accepts a key for use with B only if the key is fresh (*key freshness*) and the key is known only to A and B (*key exclusivity*).

Entity Authentication of A to B. B accepts A only if A wishes to communicate with B.

Key Confirmation: B accepts A with key K only if K is a good key to communicate with A and B has received K.

Mutual Belief in Key: B accepts A with key K only if K is a good key for use with A, and B wishes to communicate with A using key K which B believes is good for this purpose.

Boyd concludes that protocol designers should use extensional goals because they are easier to handle and implement. Roscoe, however, states that intensional goals are easier to analyze.

3 Protocol Design

Although there are many security protocols, only few papers on design methods have been published. Until recently, designing security protocols did not follow any particular rules and no formal methods were used. This led to protocols which were apparently sound but which contained subtle flaws that were often discovered years after the protocols had been published. Fortunately, the attention of researchers has now been drawn to this matter and several mechanisms have been proposed.

3.1 Design Guidelines

Abadi and Needham [AN96] list eleven general design principles. The principles are informal guidelines, but adhering to them should help to avoid some commonly made mistakes.

Principle 1: Every message should say what it means.

Principle 2: The conditions for a message to be acted upon should be clearly set out.

Principle 3: If the identity of a principal is essential to the meaning of a message, include it explicitly in the message.

Principle 4: Be clear about why encryption is being done. In a smaller recommendation, the authors also advise designers to be clear on how encryption is used, and on the meaning of encryption.

Principle 5: When a principal signs encrypted data, it should not be inferred that the principal knows the data. On the other hand, when the principal signs data and then encrypts it, it can be inferred that the content of the message is known to the principal.

Principle 6: Be clear about the properties of nonces and what they should achieve. In a smaller guideline at the beginning of the paper, the authors also recommend to be clear on how the timeliness of messages is proved, and on the meaning of temporal information in messages.

Principle 7: If a nonce is predictable, then it should be protected so that an intruder cannot simulate a challenge and later replay a response.

Principle 8: If timestamps are used as a freshness guarantee and are refering to absolute time, then the difference between all local clocks must be much less than the allowable age of the message. Furthermore, the time maintenance mechanism becomes part of the trusted computing base.

Principle 9: Recent use of keys does not imply that they are new and uncompromised.

Principle 10: If an enconding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. If the encoding is protocol dependent, then it should be possible to deduce whether the message belongs to the protocol, whether it is part of the current protocol run, and its position in the run.

Principle 11: The protocol designer should know which trust relations the protocol depends on, and why the dependence is necessary.

Liebl [Lie93] mentions several questions that a protocol designer should take into consideration:

- Does the protocol work?
- Exactly what does the protocol achieve? What beliefs are established?
- Does the protocol need more assumptions than other protocols?
- Does the protocol do anything unnecessary?
- Which encryption algorithms are used?

Additionally, protocols should have

Perfect Forward Secrecy: Disclosure of long-term secrets should not compromise the secrecy of the exchanged keys from earlier runs.

Direct Authentication: In indirect protocols, authentication is not complete until the parties prove their knowledge of a shared key by using it in the subsequent communication. So authentication is linked with key exchange and cannot exist without it.

No Timestamps: Since using timestamps introduces new difficulties and attack opportunities into protocols, they should be avoided.

3.2 Design Toolkits

Currently, only few design toolkits exist. A comprehensive toolkit is SPEAR [BdG97] and its successor SPEAR II [SH99]. SPEAR II encompasses formal specification, a modal logic analyzer for GNY logic including constructs for digital signatures and certificates, automated code generation for Java or SDL (Standard Description Language), meta-execution within the framework of SPEAR II for testing, and a performance analysis.

The protocol design stage consists of declaring the principals and defining the message passing specifications. Message passing is represented by MSC (Message Sequence Charts), message items themselves can be declared with ASN.1 (Abstract Syntax Notation One) if they need to be extracted from the message. The user can define external functions that are applied to message components, specify communication settings, and declare information specific to the modal logic analysis. The specification is done via a graphical user interface.

Security analysis deals with the progressions of beliefs and growth of possession sets during the run of the protocol. It is able to determine the degree of redundancy in the protocol, the type of authentication achieved (one-way or n-way), check for information that is sent both in plain text and ciphertext, and examine and enumerate possible replay attacks. Additionally, fail-safe analysis is incorporated.

Elaborate performance analysis is used which can subject the protocol to differing conditions and stresses, can keep track of the number of messages and their sizes, and can decide whether a protocol is optimal w.r.t. the number of messages and rounds. The impact of different algorithms like encryption methods can be evaluated, and the performance under concurrent conditions can be gauged. With the SDL output, the protocol can also be analysed by other tools specifically designed for the analysis of distributed protocols, like SPECS II or Geode.

Meta-execution allows the designer to execute the protocol in a simulated environment under different conditions. The designer should be able to view the progression of the protocol and to set up simulated attacks and scenarios for demonstrations or further research.

Source code generation already implements most of the protocol, like network communication or encryption functions. The designer just has to fill in special functions specific to the protocol.

An altogether different approach is taken by Perrig and Song [PS00]. The authors conclude from the numerous mistakes found in well-known protocols that it is best if the design is completely automated. They propose a system where the designer just has to specify the desired security properties of the protocol and the system requirements, and an optimal protocol is generated automatically.

The system requirements consist of metric functions specifying the "cost" of operations like encryption or nonce generation. The protocol generator then searches all possible protocols (automatically generating all possible message exchanges) whose costs are below a given threshold. These protocols are sorted according to their costs and then analyzed by a protocol screener to filter out the flawed ones, until a correct protocol is found. The output of the whole tool is the protocol with the least cost according to the given metrics.

4 Hardware Approaches

Apart from software protocols, some papers propose tamper-resistant hardware devices as a means to establish trust [SW99], [WSB99], [Yee94]. In Norway, the NSK cryptochip was developed to protect communications over telephone lines [PP96]. In [Gol96], several secure store and forward devices are described. Unfortunately, there are means to cope even with tamper-resistant hardware, as is demonstrated in [AK97].

The american standardization institution NIST has developed the FIPS 140 Standard [FIP99] which defines security requirements for cryptographic modules. It describes four levels of security, and developers of such modules can get their programs and/or devices certified for a given security level. Each level builds upon the previous one and adds additional security mechanisms.

Level 1: This is the lowest level of security. It does not require any special physical security mechanisms. An example would be a PC encryption board or software cryptographic functions.

Level 2: This level adds the requirement for *tamper-evident* coatings or seals or for pick-resistant locks. All critical security parameters (CSP) are placed inside the module, which are tamper-evident. Additionally, tamper-evident seals or pick-resistant locks should be placed on covers or doors to protect against unauthorized access. Authentication must at least be role-based, which means that the module authenticates the authorization of an operator to assume a specific role and to perform a corresponding set of actions.[2]

Level 3: In this level, intruders are *prevented* from gaining access to critical security parameters. Whenever tampering is detected, the parameters are deleted (zeroized). Data ports used for entering and outputting CSPs must be physically separated from other data ports. Authentication is identity-based, so the module verifies the identity of an operator and checks whether this particular operator is authorized to assume a specific role and perform actions.[3]

Level 4: This is the highest level of security. The module is required to detect penetration from any direction and zeroize all critical security parameters. Environmental protection must be available, and excursions beyond the normal environmental range must be answered with immediate zeroization of all CSPs.

5 Software Approaches

In the following subsections, we will try to sketch the basic structure of the protocols and give an example.

[2] An example would probably be the *root* account in UNIX systems. The system just checks the role (system administrator), but not the identity of the operator.

[3] In this case, the identity of the operator is checked, and the operator may have a system administrator role associated with him or her.

In general, all protocols we have encountered so far achieve their goals at best (many are even flawed) and do not concern themselves with problems like replication, fault tolerance, or even denial-of-service attacks.

5.1 Authentication

The general aim of authentication protocols is to ensure that each principal knows he is talking to the correct peer and that the peer also knows this. To achieve this aim, the protocols generally have the following structure, where Alice initiates a communication with Bob

ID	Message	Knowledge
		A knows A $\rightarrow$ B
M1	A $\rightarrow$ B: I am A	B knows A $\rightarrow$ B B knows B $\rightarrow$ A $\Rightarrow$ B knows A $\leftrightarrow$ B
M2	B $\rightarrow$ A: I am B, I am replying to M1	A knows B knows A $\leftrightarrow$ B A knows B $\rightarrow$ A $\Rightarrow$ A knows A $\leftrightarrow$ B
M3	A $\rightarrow$ B: I am A, I am replying to M2	B knows A knows A $\leftrightarrow$ B B knows A knows B knows A $\leftrightarrow$ B
M4	B $\rightarrow$ A: I am B, I am replying to M3	A knows B knows A knows A $\leftrightarrow$ B

In this table, $X \rightarrow Y$ means that X wants to talk to Y. The "Knowledge" column denotes what each principal knows after receiving the message. Message 4 is just to ensure A that B has got M3 (that is, the protocol run has completed successfully). It is the task of the particular protocol to ensure that the knowledge can indeed be derived from the messages. It is our understanding, however, that the goal cannot be achieved in less messages than stated in the above skeleton.

The last message is often left out in protocols. Its effect can be achieved by any message from B that is linked to M3, so the first message from the following communication is perfectly adequate.

Once the identity of the communication parties has been established, they can start to exchange data. However, most of the time the main goal is to ensure secrecy to the subsequent message exchange. This leads to the request for key distribution protocols.

5.2 Key Distribution

Key distribution, also called key exchange, is generally based on authentication. The aim is to provide two principals with an encryption key that is used to encrypt the subsequent communication. Basically, the encryption scheme can be symmetrical (one secret key) or asymmetrical (public/private key pair), see Appendix A.

Asymmetric schemes implicitly provide authentication, because only one principal knows the private key and can use it as a signature. On the other hand, the association between a public key and a given principal must be correct and tamper-proof, so it requires a *public key certificate* which has to be provided by a *trusted authority*. Public keys are typically valid for a long time, so the cryptosystem must be very secure. If a private key is compromised and a new key pair has to be issued to its owner, then every node who possibly uses the old public key certificate has to be notified that the key has been changed. However, this might perhaps be alleviated if certificates are exchanged as part of the initiation of a communication.

How the private key of such a key pair is distributed to its owner is not really clear. Apparently, stations get it offline. If it is sent online, then it is encrypted with a key shared between the trusted authority and the station. It is not stated what happens if this key is compromised as well. In any case, if a key has to be encrypted with another key, then the second key used for the encryption has to be stronger than the key which is exchanged. Offline entry seems to be the best thing anyway. However, this appears to be an open issue.

There is no real protocol for the distribution of the certificates. If Alice wants to send a message to Bob, she typically gets his public key certificate from a trusted authority. In fact, as long as there is only one certificate, she might get it from anywhere, the only real requirement to Alice is that she must be able to recognize and trust the signature of the trusted authority. So she could also ask Bob. After obtaining the certificate, she uses the public key contained in it to encrypt her message to Bob. Only Bob will be able to decrypt it again, so she can be sure that only Bob can read the message she sends. If Bob needs to be sure who sent the message, Alice should sign it using her own private key.

The situation gets more complicated if Bob has to change his key due to compromisation. This problem, called *key revocation*, is quite severe because just issuing a new certificate to Bob is not enough. Every node in the system has to be informed that the old certificate is not valid anymore. Otherwise, an intruder who knows Bob's old key could impersonate him by distributing the old certificate. Therefore, the approach suggested above, that Alice asks Bob to provide his own certificate, is highly insecure. An intruder might capture and suppress the message and send the old certificate as a reply. This strongly suggests that a trusted authority should be involved into the distribution of the certificates, at least, if the intruder is capable of suppressing messages.

Symmetric schemes use just one key for encryption and decryption which is known to both principals participating in the communication. This is considered to be less secure than a public key system, so these keys are normally just used for one exchange and negotiated before the start of every data exchange. The key exchange does not require a trusted authority, although some protocols use it for security reasons.

In the two-party case, a common solution is to equip all users with a secret and a corresponding public value which serves as their certificate. After the identities have been established, each party computes a master key from his or her own secret and the public value of the peer. From this master key, a communication key is derived using a one-way function. An example for this kind of protocol is SKIP (Simple Key-management for Internet Protocols), see Appendix B.

In the three-party case with a trusted authority, every user shares a secret key with the authority and uses it to obtain a key for the communication. The obvious disadvantage of a three-party method is that the authority is a bottleneck. However, not all possible keys are equally secure, and a trusted authority can be equipped with the means to check a newly generated key for its quality, cf. [LGSN89].

5.3 Secure Login

When a user logs into a system, he has to enter a password which is checked by a trusted authority. The tendency of users to choose poor passwords leads to the necessity of creating login protocols which can nevertheless protect the user from password-guessing attacks. In [LGSN89], the problem is reduced to the ability of intruders to capture a message and do offline experimentations like a dictionary attack. In consequence, the authors propose a system where the attacker must interact with a server to check whether the guess is valid, and the server can log every incorrect guess and raise an alarm. In addition, they suggest that messages should be constructed in such a way that an attacker cannot verify if he has guessed the correct key. They also recommend that the password itself should never be used in communication, instead the client should use a one-way function to generate a key from the password.

When guessing the contents of messages, *known plaintext* is a handy aid for attackers. It means that the attacker can already predict part or all of the message before decrypting it. An example would be an address field the position and contents of which is known beforehand. The knowledge can be exploited to deduce the rest of the message, or even the key used for encryption.

However, in [LGSN89] the authors concern themselves with *verifiable plaintext*, which is more general than known plaintext. It occurs if a message contains information which is recognizable when decrypted. An example would be a message which contains some random number and its complement. An attacker does not know the random number but can try out keys until he finds one that produces two numbers whose sum is zero.

As an example, we look at the login phase of the Kerberos protocol. Here, Alice logs in and sends a plaintext request for an authentication key to the trusted authority Kas. The reply is encrypted with Ka, which is a key shared between the trusted authority and Alice and which is derived from the password [NKT00].

1. A $\rightarrow$ Kas: A, Tgs, Ta1
2. Kas $\rightarrow$ A: $\{$AuthKey, Tgs, Tk, $\{$A, Tgs, AuthKey, Tk$\}_{Ktgs}\}_{Ka}$

Although the protocol is widely used, we would like to point out a few problems (more about Kerberos can be found in Appendix B). First, an intruder can just monitor the traffic to find out when each user logs in. So we would prefer to somehow hide the identity of the user. As stated in [NKT00], password-guessing attacks are not solved by Kerberos. An intruder can store message 2 and try to decrypt it with a dictionary. Sending the message A $\rightarrow$ Kas: A, Tgs, $\{$Ta1$\}_{Ka}$ might help, because now A must know the key to initiate the conversation, and Kas can log failed guesses. Of course, an intruder can always block a legitimate user by a denial-of-service attack. Kerberos cannot handle such attacks. We believe that a better method for solving the above problems is to use a public key for the initial communication with Kas. Message 1 is A $\rightarrow$ Kas: $\{$A, Tgs, $\{$Ta1$\}_{Ka}$, N$_A\}_{K_{Kas}}$. Again, $Ta1$ is encrypted to enable Kas to detect password guesses. To avoid a plaintext attack where the intruder tries to guess the plaintext and then encrypts it to see if he is right, a random nonce N_A is added to the message. The lower bits of $Ta1$ have the same effect if the intruder cannot guess the exact value.

5.4 Virtual Private Networks

Before we conclude the paper, let us briefly look at *Virtual Private Networks* (VPNs). Their aim is to provide a secure (sub-)network on top of a larger insecure public network like the Internet. This is achieved through tunnelling methods, where connections between two members of the private network are encrypted during their transit through the public network. According to [Han97], the endpoints of the tunnel can be:

- routers
- firewalls
- dedicated VPN boxes

The idea is that a VPN client A who wishes to establish a connection to some other VPN client B connects to a VPN endpoint, e.g. a router. This router calls the VPN endpoint on the receiver's side and establishes a VPN connection. All further traffic over this connection is encrypted. The method even allows for tunnelling of non-IP based protocols.

VPNs use existing security methods for achieving their secure communication. They need authentication and key exchange protocols. They do not incorporate any new ideas, but are simply an application of security methods. Therefore, they are not of interest for this survey.

6 Conclusion

There are many different protocols for the same task. Some have been designed for specific systems or assume certain services like synchronized clocks. For our system, we will definitly need protocols for authentication and for key exchange. We must also look into secure group communication. Since W_2F is only used in a secluded area, we will not need VPNs.

As far as hardware support is concerned, W_2F should include a secure co-processor and have at least FIPS Level-1 certification.

Finally, whatever protocols we choose to employ, we must subject them to formal analysis to verify whether they achieve their goals under the system assumptions of W_2F. We will need tools which are able to deal with different intruder capabilities, so we can handle different W_2F system states.

A Cryptocraphic Methods

Since security protocols generally need encryption to achieve their goals, we will give a brief overview over the different methods used for encrypting data. Basically, one has to distinguish between *symmetric* and *asymmetric* encryption methods. A good overview —albeit in german— can be found in [Sel00]. It has been used as the basis for the following subsections. A very good english book is [Sch96].

A.1 Symmetric Encryption Schemes

Symmetric methods use one key for encryption and decryption. In consequence, both the sender and the receiver have to know the key. This is why such methods are often called *shared key systems*. Of course, the receiver must obtain the key prior to decrypting the message, and transmission of the key must be secure for the scheme to work. It the receiver compromises the key, the sender must change its key as well. This makes symmetric key schemes more problematic than asymmetric ones.

Examples of symmetric systems are

DES (Data Encryption Standard): This method encrypts blocks of 64 bits with a key of length 56 (the last 8 bits of the block are a checksum). Basically, the data is combined with the key, and then processed by *S(ubstitution)-Boxes* which map a 6-bit input to a 4-bit output and *P(ermutation)-Boxes* which swap the bits of the resulting encrypted block. This is done several times (16 rounds). Decryption is achieved with the same method in reverse order. Since the algorithm is quite simple, it can easily be implemented in hardware and is therefore very fast.

Until recently, DES fell under the export restrictions of the USA, preventing longer keys to be used. However, the 56-bit version of DES has already been cracked using a brute-force algorithm which checks all possible keys until it finds the right one. The effort necessary to launch such an attack is relatively moderate, so 56-bit DES cannot be considered to be secure. A key length of 128 bit is therefore recommended by the security community. Nevertheless, DES is a problem because it is feared that the NSA (National Security Agency) of the USA, which has been involved in the development of DES, may have left some backdoor (the S/P-Boxes are suspected here) to decrypt messages even without the key.

Triple DES: This is a variant of DES which encrypts the data with a 56-bit key, decrypts it using another 56-bit key, and then encrypts it again. This has more or less the same effect as a longer key, making a brute-force attack unfeasible.

IDEA (International Data Encryption Algorithm): The method has been developed at the ETH Zurich (Switzerland) and also encrypts data blocks of 64 bits length, but uses a 128-bit key. It avoids several drawbacks of DES. The long key makes brute-force attacks unfeasible, the algorithm does not need any S- or P-Boxes, and it only needs eight rounds. The encryption is done with a combination of XOR, addition modulo 2^{16} and multiplication modulo 2^{16}. The method is considered to be very secure, but is protected by a patent.

Rijndael: The Rijndael algorithm [DR99] has been developed quite recently and has been selected as the new Advanced Encryption Standard (AES) [FIP01] by the American NIST (National Institute of Standards and Technology). The algorithm along with several other candidates has undergone intense public scrutiny. It can use keys of 128, 192, and 256 bits to encrypt data of 128 bits per block. The algorithm replaces DES as the standard. Its selection criteria included that the algorithm should be faster than Triple DES, be freely available and have been publicly scrutinized. Further information can be obtained from the AES homepage at *http://csrc.nist.gov/encryption/aes/*.

The above methods are all *block ciphers*, that is, they encrypt the plaintext one block at a time. However, this leads to the problem that an intruder can replace blocks and thus manipulate the message without being detected. For this reason, *cipher block chaining (CBC)* has been introduced. In this method, the plaintext of a block is XOR-ed to the encrypted previous block before being encrypted itself. The first block is encrypted random data to make every ciphertext unique.

An altogether different approach is to use a *stream cipher*, where the plaintext is XOR-ed to a random sequence one bit at a time. Obviously, the quality of the encryption depends on the random generator.

A.2 Asymmetric Encryption Schemes

Asymmetric methods use a pair of keys (K_{pub}, K_{priv}) which are related to each other. Everybody may obtain the public key K_{pub}, but only the owner A of the pair knows the private key K_{priv}. The keys are chosen such that $((M)_{K_{pub}})_{K_{priv}} = ((M)_{K_{priv}})_{K_{pub}} = M$. Knowing the public key does not enable anyone to guess the private key, and there is only one private key which is the inverse of the public key.

This scheme enables any user B who knows K_{pub} to send an encrypted message to the owner A of K_{priv}. Since the private key is only known to A, it is guaranteed that nobody else can read the message. On the other hand, A may encrypt a message using K_{priv} and send it to B who applies K_{pub} to obtain the plaintext.

Since only K_{priv} is the inverse of K_{pub} and only A knows K_{priv}, B is assured that the message has been sent by A. Thus, asymmetric methods can be used to electronically sign messages. Their major disadvantage, however, is that due to the difficult computations involved, encryption takes much longer than with symmetric schemes.

Well-known asymmetric schemes are

RSA: This method, which has been named after its inventors Rivest, Shamir, Adleman [RSA78], has been the first asymmetric method to become widely popular. It is based on the hardness of factorization of large numbers into their primes. In this scheme, Alice, who wishes to generate a key pair, chooses two large prime numbers, p and q. She then computes $n = p * q$ and $r = (p-1) * (q-1)$. Now, she chooses an arbitrary number e which must not have a common divisor with r. The numbers e and n together form the public key. Additionally, Alice computes d such that $e * d = 1 \; mod \; r$. The numbers d and n form Alice's private key. If user Bob wants to send Alice a message, he splits it into blocks m which are each smaller than n and computes $Enc(m) = m^e \; mod \; n$. These encrypted blocks form the message which is sent to Alice. Alice now computes $Dec(Enc(m)) = Enc(m)^d \; mod \; n$. Since d is the inverse to e, Alice obtains the original block m.

The RSA scheme is protected by a patent in the USA (until 2000).

ElGamal: This scheme, also named after its inventor, uses discrete logarithms. The problem of computing from a given y, g, and p the x for which holds $y \equiv g^x \; mod \; p$ is extremely hard for large p (at least as hard as the factorization problem utilized by RSA). Alice creates a pair of keys by choosing a sufficiently large prime number p and some arbitrary numbers g and x and computes $y \equiv g^x \; mod \; p$. The numbers p, g, and y form the public key, x is kept secret. When Bob wants to send a message, he parts the message into blocks m which are smaller than p. Then he chooses an arbitrary number k which has no common divisor with $p-1$ and computes the two numbers $a \equiv g^k \; mod \; p$ and $b \equiv y^k m \; mod \; p$, which he sends to Alice. The number k must be kept secret by Bob. Upon reception, Alice computes b/a^x to get the plaintext m.

The main disadvantage of asymmetric schemes is that the execution of the algorithm is very time-consuming and that secure keys are much longer than in symmetric schemes (e.g., RSA needs at least 1024 bits). Additionally, it is hard to change the public key, since most users will download it only once and then store it locally. On the other hand, the schemes are scalable because the effort necessary to break the method grows exponentially with the size of the chosen numbers, so whenever computers get fast enough to break a key one can always use larger numbers to make brute-force attacks unfeasible again.

A.3 Hash Functions

In contrast to encryption methods, which aim to keep the contents of messages secret, *hash functions*, which are also called *message digest functions*, try to guarantee the integrity of messages. The idea is to append to the message a short sequence of bytes which has been computed from the message itself. The function used for obtaining the hash value should be easy to compute, reasult in a small hash value of fixed length independend of the original message length, and it should be impossible to construct a message that has the same hash value than a known hash value. In fact, it is often demanded that it should be unfeasible to find two messages that have the same hash value (this is called a *collision*).

The sender of a message computes its hash, sends the message to the receiver and uses some *secure* way to transmit the hash of the message to the receiver. The receiver also computes the hash of the message and compares it to the hash stated by the sender. If the two values match, then the message has not been tampered with.

MD5: *Message Digest 5* is the last of a series of message digest functions developed by Ron Rivest (the others are MD2 and MD4). It computes hash values of 128 bits length, has been widely used, is very fast, but unfortunately has some weaknesses that allow to find collisions. Therefore, its use is not recommended.

SHA-1: The *Secure Hash Algorithm* has been developed by the NSA and is considered as one of the best currently known methods. It computes hash lengths of 160 bits and operates on 512 bits of the message with bit shuffling, addition, bit shifting, and some non-linear function.

RIPEMD-160: This algorithm has evolved from an EU project and is very similar to SHA-1. Its hash value also has 160 bits, although the authors have also described variants that use 128, 256, and 320 bits. The method also uses 512 bit blocks and uses shifting, addition and a non-linear function which is similar to that used in SHA-1.

Currently, SHA-1 and RIPEMD-160 are the best hash functions available. Their performance is comparable, although SHA-1 is slightly faster than RIPEMD-160. None is protected by any patents.

A.4 Signatures

As we have seen in the previous section, hash functions guarantee the integrity of the message, provided that the hash value is transmitted to the receiver in a secure way. A signature normally does a lot more: it binds the sender (who has signed a message) to the message. This involves guaranteeing the authenticity of the sender, the integrity of the message, and the connection between the message and its signature.

Fortunately, there is a simple method to achieve all requirements to electronic signatures which is based on asymmetric key encryption in combination with message digest functions: the sender Alice simply computes the hash value of the message, encrypts this value with her private key, and appends the hash to the plaintext message. The receiver Bob decrypts the hash and compares it with his own computation of the hash of the message. This guarantees Bob that the message has not been tampered with (the hash values are equal), that Alice has been the sender of the message (only Alice could encrypt the hash value with her private key), and that Alice has sent this particular message (Alice has sent the correct hash value). In order to guard herself against possible attacks or misuse of her message (e.g., sending the same bank transaction request twice), Alice should also include a timestamp or some other unique value into her documents before she signs them. If the message should also be secret, Alice first appends her signature to the plaintext message and then encrypts the complete message (including her signature) with Bobs public key.

The american standard *DSS (Digital Signature Standard)* [FIP00] describes a method which uses SHA-1 for computing the hash value and a public key cryptosystem to compute two values from the hash and the private key. These are transmitted as the signature. The receiver can verify the signature by using the public key.

In general, signatures or hash functions which are based on shared secret keys are called *Message Authentication Codes* (MACs). Since the value is based both on the message and on a secret which is only known to the sender and the receiver, an intruder cannot simply intercept the message, change it, and forge the hash function for it. What is more, he cannot even try to change the message until it has the same signature or hash function as the original message, because he does not know how to compute it. Obviously, MACs guarantee both the integrity of the message as well as the authenticity of the sender, provided that the secret key is only shared between the sender and the receiver.

B Some Protocols

The most important attack types on protocols are

Denial of Service (DoS): This attack tries to cause the server to deny its service to a legitimate user. A typical way to achieve this goal is to flood the server with bogus messages. A more subtile method would be to replace the message from a legitimate user with a fake message that causes the server to abort the protocol run, or to send additional messages with this result. Normally, denial-of-service attacks are not considered in security analyses, possibly because they are very hard to prevent, especially if the intruder is assumed to have full control over the network.

Message Replay: The intruder takes a message sent by a legitimate user (Alice) during a previous protocol run and replays it in a current run. The aim is to reuse information of the old message so that the current communication partner (another legitimate user Bob) accepts it as a valid message in the current run. Bob should possibly be tricked into believing he is talking to Alice instead of the intruder. In any case, he should conclude that the intruder knows more information (secrets) than is the case.

Parallel Session: Here, the intruder opens a session with two legitimate users (Alice and Bob) at the same time and uses the messages from Alice to reply to challenges from Bob. The effect is much the same as in the message-replay attack, Bob will believe he is talking to Alice instead of the intruder. However, the intruder can use the parallel session with Alice to dynamically react to a challenge from Bob (he challenges Alice and sends her reply to Bob). The attack type is also called *man-in-the-middle* attack.

Timing Attack: Any protocol that uses timestamps as nonces is susceptible to suppress-replay attacks, where the message from a node with an overly fast clock is suppressed by the attacker until the timestamp becomes valid and is then replayed.

In the following sections, we describe some commonly used security protocols. To allow comparison of different algorithms, we will list the number of messages they need. Generally, we feel that an algorithm which uses few messages presents less opportunity for attacks than an algorithm with many messages, although of course this also depends on the nature and contents of the messages. See [Gon95] for lower bounds on the number of messages required for authentication algorithms. We will also list how many encryption operations each algorithm requires. Encryption operations can be exploited to launch denial-of-service attacks.

B.1 CCITT X.509

Goals: Authentication

Messages: 4

Encryption: 2 (2)

The *CCITT X.509* authentication protocol, which is a standard, has been analysed in [GS91]. The paper only analyses the "two-way strong authentication" mode of the standard (the other modes would be one-way and three-way). This mode of the protocol needs two clients A and B and a certification authority (CA) which signes certificates for A and B. The certificate for a user A which has been issued by I basically contains the names of A and I, a duration, the public key of A, and it is signed by I, i.e., $C_A = (I, \delta, A, P_A)_I$. The clients exchange signed tokens which contain the name of the receiver, a timestamp and duration, a random number generated by the sender, and optionally some signed data encrypted with the public key of the receiver, so $G_A = (t_A, R_A, B, sgnData, E(p_B, encData))_A$ and $G_B = (t_B, R_B, A, R_A, sgnData, E(p_A, encData))_B$. The protocol exchanges the following messages:

1. A $\rightarrow$ CA: A, B
2. CA $\rightarrow$ A: C_A, C_B
3. A $\rightarrow$ B: C_A, G_A
4. B $\rightarrow$ A: G_B

Message 1 is a request for certification. In Message 2, the CA forwards to A both A's and B's certificates, which are both signed by the CA and which have a limited validity (due to the lifetime δ). A checks the signature of the CA on C_B and if it succeeds, it sends its own certificate along with its token G_A to B in Message 3. B checks the signature of the CA on C_A, then checks A's signature on G_A. If the checks succeed, B processes the contents of the token. B replies in Message 4 with its own token G_B, which is likewise checked by A. Note that this message is linked with Message 3 by including A's random number R_A into the reply.

According to [GS91], the goals of CCITT X.509 are:

- After a run of the protocol, both A and B should believe they have a valid public key of the other.
- Both A and B should believe that the token they have received has recently been sent by their peer.
- Both A and B should believe that the are legitimate receivers of the respective token.

- Optionally, they should believe that their peer believes the *sgnData* part of the token.

- Optionally, they should believe in the mutual secrecy of part of the token.

The authors of [GS91] can prove the first four goals, but not the last one. The reason is that neither A nor B provide any knowledge about the value of the secret.

Like Kerberos, CCITT X.509 also depends on synchronized clocks. Therefore, the remarks concerning the vulnerability of Kerberos in case of clock faults also hold for X.509.

B.2 Diffie-Hellman

Goals: (Shared) Key Exchange

Messages: 2

Encryption: 0

The Diffie-Hellman [DH76] key exchange protocol has been in use for some decades now. It can compute a shared secret key between two parties, see [Sch96, p. 513ff.] for a good overview. The protocol assumes that two parties select a large prime n and some value g that is primitive mod n. These two values are public knowledge. To establish a common shared key, Alice resp. Bob each select a large random number x resp. y and exchange the following information:

1. A $\rightarrow$ B: X $= g^x \bmod n$

2. B $\rightarrow$ A: Y $= g^y \bmod n$

Since $k = Y^x = X^y = g^{xy}$, Alice and Bob now share a key k than cannot be computed by anyone else. The method relies on the fact that it is a lot easier to compute the exponent than it is to calculate the discrete logarithm in a finite field. However, n should be chosen well, it must be large and $(n-1)/2$ should also be a prime.

The algorithm can easily be extended for communication among groups by adding more rounds, see [Sch96, p. 514]. Better algorithms that minimize the number of rounds and the amount of computations can be found in [STW96].

Note that the algorithm is open to parallel session attacks, since Alice and Bob are not authenticated.

B.3 Kerberos

Goals: Authentication, Accounting, Authorization

Messages: 2 + 4

Encryption: 2 + 5

One of the most well-known protocols is *Kerberos* [NKT00], which is a shared key protocol. Its goals are authentication, accounting and authorization. In [BP98], Kerberos Version IV is described and analysed. Kerberos consists of two principals and two trusted third parties, the Kerberos Authentication Server (Kas) and the Ticket Granting Server (Tgs). Upon login, a principal obtains an authorization key `AuthKey` from Kas, which has a limited lifetime. After the key expires, the principal is automatically logged out. In order to use the remote service of some principal B, a principal A has to ask the Tgs for a service key `ServKey`, which has again a limited lifetime. With this key, A requests a service from B. The messages necessary for completing the protocol are as follows:

1. A $\rightarrow$ Kas: A, Tgs, Ta1
2. Kas $\rightarrow$ A: $\{$AuthKey, Tgs, Tk, $\{$A, Tgs, AuthKey, Tk$\}_{Ktgs}\}_{Ka}$
3. A $\rightarrow$ Tgs: $\{$A, Tgs, AuthKey, Tk$\}_{Ktgs}$, $\{$A, Ta2$\}_{AuthKey}$, B
4. Tgs $\rightarrow$ A: $\{$ServKey, B, Tt, $\{$A, B, ServKey, Tt$\}_{Kb}\}_{AuthKey}$
5. A $\rightarrow$ B: $\{$A, B, ServKey, Tt$\}_{Kb}$, $\{$A, Ta3$\}_{ServKey}$
6. B $\rightarrow$ A: $\{$A, Ta3+1$\}_{ServKey}$

The first two messages are issued only once, during the login phase. The `AuthKey` is valid for a given time, after which A is automatically logged out. Messages 3 and 4 are exchanged whenever A needs to use the service of some remote principal B. A gets a `ServKey`, which is again valid for a certain time, and a ticket encrypted with B's key. With this information, A contacts B to request the service and B sends back an acknowledgement (messages 5 and 6). Kerberos uses the Data Encryption Standard (DES) for encryption, a symmetric cryptographic method using just one key for both en- and decryption.

As pointed out in [BP98], leakage of the `AuthKey` results in compromisation of the `ServKey`. This can be a problem if the lifetime of the `ServKey` is longer than that of the `AuthKey` and the latter is obtained after it has expired. The spy can then decrypt message 4 and obtain the server ticket to request services in the name of A. As a solution, the authors propose to limit the lifetime of the `ServKey` to that of the `AuthKey`.

Obviously, Kerberos requires synchronized clocks to make use of the timestamps and lifetimes sent with the messages. In [Gon92], it is shown that Kerberos is vulnerable to attacks based on faulty clocks. The scenario describes a suppress-replay attack during which the post-dated Message 1 from a client with a fast clock is suppressed, kept until the timestamp has been reached, and then replayed, prompting the server to accept the message as valid.

Kerberos V enhances version IV by fixing some known deficiencies [BM91]. But it also introduces new problems: Tickets can now be forwarded, allowing clients to use services that are depending on other services. However, this feature causes the problem of cascading trust: A may trust B, and B may trust C, but this does not imply that A trusts C. However, since the originator of a forwarded message is unknown, A may get requests from C without knowing it.

B.4 Needham-Schroeder

Goals: Authentication

Messages: 7

Encryption: 3 (2)

The *Needham-Schroeder protocol* uses public keys and a trusted authority is required to send these public keys to the users. The original protocol had a flaw which has been pointed out by Lowe [Low96]. Another flaw has been found by Meadows [Mea96]. The following version already incorporates the fix proposed by Lowe, which also repairs the flaw found by Meadows. A stronger version using authentication has been suggested by Boyd [Boy97].

1. A $\rightarrow$ S: B
2. S $\rightarrow$ A: $\{\mathrm{K}_B, \mathrm{B}\}_{K_S^{-1}}$
3. A $\rightarrow$ B: $\{\mathrm{N}_A, \mathrm{A}\}_{K_B}$
4. B $\rightarrow$ S: A
5. S $\rightarrow$ B: $\{\mathrm{K}_A, \mathrm{A}\}_{K_S^{-1}}$
6. B $\rightarrow$ A: $\{\mathrm{N}_A, \mathrm{N}_B, \mathrm{B}\}_{K_A}$
7. A $\rightarrow$ B: $\{\mathrm{N}_B\}_{K_B}$

Alice requests Bob's public key in message 1 and receives it (signed by the trusted authority) in message 2. She then sends a nonce and her name to Bob, who in turn obtains Alice's public key in messages 4-5. Bob then replies with Alice's nonce, his own one, and his name, in message 6, and Alice completes the protocol by returning Bob's nonce in message 7. After the run, both parties are certain about the identity of their peer.

We note that the algorithm strongly depends on the security of the link to the CA. Assume that there is an intruder I sitting between S and A. Then I can send any $\{K_B, B\}_{K_S^{-1}}$ message S has ever sent and suppress recent messages from S. If I knows some old key K_B, it can send that key to A in message 2 and is then able to read everything sent by A. Likewise, it can give B an old known key K_A in message 5 and is then able to read everything sent by B. This will continue until a message from S containing the current keys comes through.

The algorithm in fact consists of two parts, the retrieval of the public keys from the CA in messages 1-2 and 4-5, and the authentication in messages 3 and 6-7. Message 3 can be sent by anyone, but if the keys used are fresh, then the sender can be assured that only B can read the message and the nonce N_A. Message 6 can only have been sent by B because it includes N_A, and the nonce N_B contained in it can only be read by A. After this message, A knows that B has received the communication request and is willing to communicate with A. Message 7, finally, can only have come from A, so now B knows that it really is A that wishes to talk.

I presume that the nonces should be included in the following data exchange as well if the peers want to be sure that the exchange is linked to this particular session.

B.5 Neuman-Stubblebine

Goals: Authentication

Messages: 4 + 3

Encryption: 4 + 3

The *Neuman-Stubblebine protocol* [NS93] has been analysed in [HLL+95] and two attacks have been described. The protocol provides mutual authentication and uses timestamps as nonces. It needs less messages than protocols using nonces based on random number, but does not depend on synchronized clocks like timestamp-based protocols. It only requires that all servers have a local clock. The protocol is parted into an *initial authentication* during which a session key is established, and a *subsequent authentication* in which communication is started. In the following description, N_X is a nonce generated by principal X, K_{XY} is a secret key shared between the principals X and Y, and T_X is the expiration time suggested by principal X. This is the corrected version of the original protocol.

1. A $\rightarrow$ B: A, N_A
2. B $\rightarrow$ S: B, $(A, N_A, N_B, T_B)_{K_{BS}}$
3. S $\rightarrow$ A: $(B, N_A, K_{AB}, T_B)_{K_{AS}}$, $(A, K_{AB}, T_B)_{K_{BS}}$, N_B
4. A $\rightarrow$ B: $(A, K_{AB}, T_B)_{K_{BS}}$, $(N_B)_{K_{AB}}$

In the initial authentication, Alice sends a request to Bob in message 1. She includes a nonce to identify answers to her request. Bob, who like all principals has a secret key K_{BS} for communication with the authentication server S, sends Alice's name and nonce, together with an expiration time and a nonce of his own, to the authentication server in message 2. The server generates a new session key K_{AB} and forwards this key to Alice in message 3, together with a ticket for Bob. In message 4, Alice sends the ticket to Bob, together with his nonce which she encrypts with the new session key. This assures Bob that message 4 came from Alice.

1. A $\rightarrow$ B: N_{A1}, (A, K_{AB}, $T_B)_{K_{BS}}$
2. B $\rightarrow$ A: $(N_{A1}, N_{B1})_{K_{AB}}$
3. A $\rightarrow$ B: $(N_{B1})_{K_{AB}}$

In the subsequent authentication, Alice requests the communication by sending Bob a new nonce and his ticket. Bob replies by encrypting Alice's nonce with the session key (this assures Alice that message 2 came from Bob) and by sending a nonce of his own. In message 3, Alice encrypts this nonce with the session key, thus proving her identity to Bob.

B.6 SKIP

Goals: Shared Key Exchange (together with data exchange)

Messages: 1

Encryption: 2

A *Simple Key-Management for Internet Protocols (SKIP)* is proposed in [AP95]. The protocol is intended for connectionless datagram communication and like the protocols in [LL95] it assumes that every user I has secret i and a public key $g^i \ mod \ p$, so two users I and J share the secret $g^{ij} \ mod \ p$ from which they can compute a common master key K_{ij}. When Alice wants to send Bob a message, she generates a random packet key K_P, encrypts her message with it, and sends the encrypted message together with K_P, which is encrypted with the master key K_{AB}. The protocol requires only one message, which already contains the data that should be sent:

1. A $\rightarrow$ B: A, B, $(K_P)_{K_{AB}}$, $(\text{Message})_{K_P}$

Only Bob can compute the shared key K_{AB} and can thus extract the packet key K_P. Alice is free to change the session key as often as she likes. The authors suggest that the shared secret key K_{AB} should be changed as well. They propose not to use K_{AB} itself, but to use keys $K_{ABn} = h(K_{AB}, n)$, where h is a suitable hash function and n is a monotonically increasing value that is transmitted with every message.

The protocol can be extended to multicast communication, where a *group interchange key (GIK)* takes the role of the shared secret key. The GIK is managed by the owner of the group, and every new member uses the unicast version of SKIP to send a request to the owner, which checks the access rights of the new member and then sends it the GIK. As in the unicast version, every member of the group uses random keys to encrypt their packets, which makes the protocol harder to analyse.

B.7 SPLICE/AS

Goals: Authentication

Messages: 2 + 6

Encryption: 2 + 5

As an alternative for Kerberos, which does not scale well due to the large amount of shared keys that have to be managed, the *SPLICE/AS protocol* has been proposed for large systems. It is very similar to Kerberos and also provides authentication, but achieves it with a public key cryptosystem. In [HC95], the protocol is analyzed and two flaws of the original proposal are pointed out and corrected. The protocol has two parts, a login and a service request. In the following description, C is the client, S the server, AS the authentication server, PK_X the public key of user X, SK_X the private key of user X.

1. C $\rightarrow$ AS: C, $(\mathrm{C}, \mathrm{AS}, \mathrm{nonce})_{PW_C}$
2. AS $\rightarrow$ C: AS, $(\mathrm{C}, \mathrm{AS}, \mathrm{nonce}, \mathrm{PK}_C, \mathrm{SK}_C, \mathrm{PK}_{AS})_{PW_C}$

During the login, the client uses her password to (symmetrically) encrypt the request to the AS. The AS replies with a public/private key pair for the client and with its own public key. All following service requests use an asymmetric encryption scheme and require the following messages:

1. C $\rightarrow$ AS: C, S, nonce_1
2. AS $\rightarrow$ C: AS, $(\mathrm{AS}, \mathrm{C}, \mathrm{nonce}_1, \mathrm{S}, \mathrm{PK}_S)_{SK_{AS}}$
3. C $\rightarrow$ S: C, S, $(\mathrm{C}, \mathrm{timestamp}, \mathrm{life}, (\mathrm{nonce}_2)_{PK_S})_{SK_C}$
4. S $\rightarrow$ AS: S, C, nonce_3
5. AS $\rightarrow$ S: AS, $(\mathrm{AS}, \mathrm{S}, \mathrm{nonce}_3, \mathrm{C}, \mathrm{PK}_C)_{SK_{AS}}$
6. S $\rightarrow$ C: S, C, $(\mathrm{S}, \mathrm{nonce}_2{+}1)_{PK_C}$

The client gets the public key of the server from the authentication server (messages 1 and 2), then requests the service in message 3. The server obtains the public key of the client in message 4 and 5, and then replies to the request in message 6. The advantage of this protocol is that the authentication server only has to manage one pair of public/private keys for each user, whereas in Kerberos every service request requires a new shared key. Another advantage is that service requests in SPLICE/AS do not need the AS if the client and the server do already know the public keys of their peers.

B.8 TLS

Goals: Authentication, Secrecy

Messages: 8 (6 without optional ones)

Encryption: 2 (1 without optional ones)

The *Transport Layer Protocol (TLS)* [DA99] is a descendant of SSL 3.0 and is used for Internet security. In [Pau97], it is analysed with the Isabelle theorem prover. TLS has been designed to protect transmissions. Its goals are authentication and secrecy. The protocol allows two parties, Alice and Bob, to exchange nonces and compute session keys from them. With the final message, both parties should be sure that they are communicating with each other, that they have both agreed on the same set of critical parameters (like the nonces), and that the following transmission will be secret. Paulson states that the protocol is very complicated and has many options. He has chosen the following version of the protocol for his analysis:

1. A $\rightarrow$ B: A, N_A, Sid, P_A
2. B $\rightarrow$ A: N_B, Sid, P_B
3. B $\rightarrow$ A: certificate(B, K_B)
4. A $\rightarrow$ B: certificate(A, K_A) (optional)
5. A $\rightarrow$ B: $(\mathrm{PMS})_{K_B}$
6. A $\rightarrow$ B: $(\mathrm{Hash}(\mathrm{N}_B, \mathrm{B}, \mathrm{PMS}))_{K_A^{-1}}$ (optional)
7. A $\rightarrow$ B: $(\mathrm{Finished})_{clientK(N_A,N_B,M)}$
8. B $\rightarrow$ A: $(\mathrm{Finished})_{serverK(N_A,N_B,M)}$

In Message 1, Alice sends a *client hello* message to B, containing her name, a nonce N_A, an arbitrary session identifier Sid, and her preferences for encryption and compression P_A. Bob replies with a *server hello* message containing his nonce, the session key, and his own preferences. Bob also sends his public key certificate in Message 3. Optionally, Alice can send her own certificate to Bob to authenticate herself. Alice then computes a *Pre-Master Secret (PMS)* and sends it to Bob encrypted with his public key in Message 5. Optionally, Alice may also send a hash of the important data, signed with her own private key. Both parties use this value and the nonces to compute a *master secret* M and calculate session keys from the nonces and M. Before sending application data, both parties exchange *Finished* messages (7 and 8) to confirm the details of the handshake. Tampering will be noticed at this point.

Paulson's analysis in [Pau97] comes to the result that the basic protocol goals, which are

1. The peer's identity can be authenticated.
2. The negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.
3. No attacker can modify the negotiation communication without being detected by the parties.

can all be proved. However, he suggests to change Message 5 to $(\mathrm{A}, \mathrm{PMS})_{K_B}$ to ensure B that the PMS came from Alice. For Message 6, the TLS specification states that the server's name and nonce should be hashed. However, analysis suggests that the pre-master-secret is also necessary.

B.9 Wireless Protocols

There are also some protocols which are written specifically for wireless systems. A wireless environment is special because eavesdropping is a lot easier and cannot be detected. In addition, mobile computers typically have less computing power, are not always online, and may change their position. This makes it harder to provide authentication and secrecy to the users. Note that normally a "wireless" system denotes a system where there are both mobile and stationary principals. The stationary computers are connected to a wired network, and only some of them, so-called *base stations* provide access points to the wireless part of the network.

In [BM98], several key establishment protocols for a wireless system are surveyed and flaws are pointed out. An *authentication and key exchange protocol* is presented in [AD94]. The authors state that a wireless network is often added to an existing wired one and should be seamlessly integrated. To achieve this goal, security mechanisms have to concentrate on securing the wireless link itself, instead of aiming for end-to-end security. This ensures that adding wireless components and the security mechanisms specifically tailored for them will not require modifications on most nodes on the wired network. Dedicated base stations are the link between the wireless and the wired network, and mobile stations communicate directly with them.

The protocol suggested by the authors needs three messages:

1. M $\rightarrow$ B: Cert(M), N_M, alglist
2. B $\rightarrow$ M: Cert(B), $\{RN_1\}_{P_M}$, selalg, $\{h(\{RN_1\}_{P_M}$, selalg, N_M, alglist$)\}_{P_B^{-1}}$
3. M $\rightarrow$ B: $\{RN_2\}_{P_B}$, $\{h(\{RN_1\}_{P_M}$, $\{RN_2\}_{P_B}\}_{P_M^{-1}}$

In message 1, the mobile sends its public key certificate, a nonce, and a list of possible shared-key algorithms to the base station. The base replies with its own certificate, a random number, the selected algorithm, and its signature on a hash of these three items plus the original list of algorithms. This ensures the mobile that the message has come from the base and is in reply to its own message. In message 3, the mobile sends another random number RN_2 and its own signature on both RN_1 and RN_2. The signature serves for the base as an authentication of the mobile. If the protocol run completes, then the shared key used for the subsequent data transfer is computed by $RN_1 \oplus RN_2$. This ensures that an intruder would have to compromise the keys of the base and the mobile to obtain the key for the data transmission.

The authors have analyzed their protocol using BAN logic and have proven that at the end of the protocol, both the mobile and the base believe that the shared key they have computed is a good key for communication between them.

Lin and Harn [LH95] focus on *mutual authentication and key exchange* in wireless networks with roaming mobile stations. They claim that established standards like GSM and DECT do not incorporate sufficient protection against masquerading and propose new protocols which are better suited to the needs of roaming stations. Their protocols provide mutual authentication between the roamer and the visited network and do not require the network to store sensitive data which might be compromised. The protocols use public key encryption only sparingly and ensure that the roamer, which is a mobile station without much computing power, only has to perform simple computations.

B.10 Other Protocols

Many papers propose protocols to solve special problems. Their basic ideas are often used by larger protocols.

In [LL95], *five protocols which combine mutual authentication with secret key exchange* are described. There must be a trusted authority to publish some global data (like public keys, prime numbers p and q which have certain properties) and to issue private data (like the secret key) to the users. So all users know the public key $P_i = g^{-S_i}$ of any user i, and only user i knows his or her secret key S_i. Apart from this initial information exchange between the user and the trusted authority, it is not needed. The protocols are based on the intractability of computing the discrete logarithm over a finite field. They use random numbers to compute the secret key. For example, the first protocol, which is the simplest but which needs encryption to work, exchanges the following messages:

1. A $\rightarrow$ B: $E_K(R_A)$, where R_A is a random number.
2. B $\rightarrow$ A: $E_{K_B}(A, R_B)$, where R_B is again a random number and the key K_B has been computed from R_A and K with $K_B = K \oplus R_A$.[4]
3. A $\rightarrow$ B: $E_{K_A}(R_A)$, where $K_A = K \oplus R_B$.

The key K used in encrypting the first message is computed by both parties from a master key $K_M = P_A^{S_B} = P_B^{S_A} = g^{-S_A * S_B}\ mod\ p$ using a suitable one-way function. Since only user A knows the secret key S_A and only B knows S_B, both users can be certain that the random numbers must have come from the correct peer. Only B could have known how to compute K_B in message 2, and only user A was capable to compute the right K_A for message 3. So after the message exchange, both users know that the random numbers must have come from the correct peer, and can compute a session key $K_S = R_A \oplus R_B$.

Gong [Gon89] presents an authentication protocol that uses one-way functions to ensure secrecy. The protocol requires the existance of a trusted authority. In the following protocol, P_A and P_B are Alice's and Bob's passwords which they share with the server S, N_i is a nonce generated by principal i, and f and g are two one-way functions.

1. A $\rightarrow$ B: A, B, N_A
2. B $\rightarrow$ S: A, B, N_A, N_B
3. S $\rightarrow$ B: N_S, f(N_S, N_B, A, P_B)$\oplus$(k, H_A, H_B), g(k, H_A, H_B, P_B)
4. B $\rightarrow$ A: N_S, H_B
5. A $\rightarrow$ B: H_A

In message 1, Alice initiates a conversation with Bob. Bob sends the request to the server in message 2. The server generates a nonce N_S and computes

[4] The $\oplus$ denotes the bit-wise exclusive-or operation.

$(k, H_A, H_B) = f(N_S, N_A, B, P_A)$. In the triple, k is the secret to be shared between Alice and Bob, H_A and H_B are handshake numbers. The server then computes f and g and sends it back to Bob in message 3. Bob computes $f(N_S, N_B, A, P_B)$ to retrieve (k, H_A, H_B) and computes g to check that the message has not been tampered with. Since P_B has been used in the computation, Bob can be sure that the message has come from the server. The message must also be fresh because it included N_B. Bob now sends N_S and H_B to Alice in message 4. Alice now computes $f(N_S, N_A, B, P_A)$ to get (k, H_A, H_B). If her H_B matches the one Bob sent her, she deduces that she has received k correctly. She acknowledges this in message 5 by sending H_A.

B.11 Comparison

The following table compares the protocols we have described in the previous sections. It shows the design goals of the protocols.

Protocol	Acc	Authen	Author	Integ	Secr
CCITT X.509 [GS91]		Y			
Diffie-Hellman					
Kerberos [NKT00], [BP98]	Y	Y	Y		
Needham-Schroeder		Y			
Neuman-Stubblebine		Y			
SKIP		Y			
SPLICE/AS		Y			
TLS [DA99]		Y			Y

This table compares the requirements of the protocols. Here, TA stands for trusted authority, SS for shared secret, CS for clock synchronization, ES for encryption scheme.

Note that the Neuman-Stubblebine protocol uses timestamps, but does not require synchronized clocks.

Protocol	TA	SS	CS	ES
CCITT X.509	Y	N	Y	asym.
Diffie-Hellman				
Kerberos	Y	N	Y	sym.
Needham-Schroeder				asym.
Neuman-Stubblebine	Y	N	N	asym.
SKIP	N	Y	N	sym.
SPLICE/AS	Y	N	N	asym.
TLS	N	N	N	asym.

References

[AD94] Ashar Aziz and Whitfield Diffie. Privacy and authentication for wireless local area networks. *IEEE Personal Communications*, First Quarter:25–31, 1994.

[AK97] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. In *Security Protocols, 5th International Workshop*, LNCS 1361, pages 125–136, April 7–9, 1997.

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[AP95] Ashar Aziz and Martin Patterson. Simple Key-management for Internet Protocols (SKIP). In *Proceedings of INET'95*, 1995.

[BdG97] J. P. Bekmann and P. de Goede. Multi-dimensional security protocol engineering using SPEAR. Technical Report CS97-20-00, Department of Computer Science, University of Cape Town, South Africa, October 2, 1997.

[BM91] Steve Bellovin and Michael Merritt. Limitations of the Kerberos authentication system. In *USENIX Winter Conference*, 1991.

[BM98] Colin Boyd and Anish Mathuria. Key establishment protocols for secure mobile communications: A selective survey. In Colin Boyd and Ed Dawson, editors, *3rd Australasian Conference, ACISP'98*, LNCS 1438, pages ??–??, July 13–15, 1998.

[Boy97] Colin Boyd. Extensional goals in authentication protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers University, September 1997.

[BP98] Giampaolo Bella and Lawrence C. Paulson. Kerberos Version IV: Inductive analysis of the secrecy goals. In Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, and Dieter Gollmann, editors, *5rd European Symposium on Research in Computer Security, ESORICS'98*, LNCS 1485, pages 361–375, September 16–18, 1998.

[DA99] T. Dierks and C. Allen. The TLS protocol, version 1.0, January 1999. RFC 2246.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael, September 1999.

[FIP99] FIPS. Security requirements for cryptographic modules, FIPS PUB 140-2, 1999.

[FIP00] FIPS. Digital Signature Standard (DSS), FIPS PUB 186-2, January 27, 2000.

[FIP01] FIPS. Advanced Encryption Standard, FIPS PUB 197, November 2001.

[Gol96] David M. Goldschlag. Several secure store and forward devices. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security, CCS'96*, pages 129–137, March 14–15, 1996.

[Gon89] Li Gong. Using one-way functions for authentication. *ACM Computer Communication Review*, 19(5):8–11, October 1989.

[Gon92] Li Gong. A security risk of depending on synchronized clocks. *ACM Operating Systems Review*, 26(1):49–54, January 1992.

[Gon95] Li Gong. Efficient network authentication protocols: Lower bounds and optimal implementations. *Distributed Computing*, 9(3):131–145, 1995.

[GS91] Klaus Gaarder and Einar Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal Of Cryptology*, 3:81–98, 1991.

[Han97] B. Hancock. Virtual private networks: what, why, when, where and how. *Network-Security*, pages 8–11, August 1997.

[HC95] Tzonelih Hwang and Yung-Hsiang Chen. On the security of SPLICE/AS – the authentication system in WIDE Internet. *Information Processing Letters*, 53(2):97–101, January 1995.

[HLL+95] Tzonelih Hwang, Narn-Yih Lee, Chuan-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen. Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letters*, 53(2):103–107, January 1995.

[LGSN89] T. Mark A. Lomas, Li Gong, Jerome H. Saltzer, and Roger M. Needham. Reducing risks from poorly chosen keys. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 14–18, December 1989.

[LH95] Hung-Yu Lin and Lein Harn. Authentication protocols for personal communication systems. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 256–261, August 1995.

[Lie93] Armin Liebl. Authentication in distributed systems: A bibliography. *ACM Operating Systems Review*, 27(4):31–41, 1993.

[LL95] Chae Hoon Lim and Pil Joong Lee. Several practical protocols for authentication and key exchange. *Information Processing Letters*, 53(2):91–96, January 1995.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Tiziana Margaria and Berhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 2nd International Workshop TACAS'96*, LNCS 1055, pages 147–166. Springer Verlag, March 27–29, 1996.

[Mea96] Catherine A. Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In Elisa Bertino, Helmut Kurth, Giancarlo Martella, and Emilio Montolivo, editors, *4rd European Symposium on Research in Computer Security, ESORICS'96*, LNCS 1146, pages 351–364, September 25–27, 1996.

[NKT00] Clifford Neuman, John Kohl, and Theodore T'so. The Kerberos network authentication service (V5), March 10, 2000. INTERNET-DRAFT, Expires September 10, 2000.

[NS93] B. Clifford Neuman and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *ACM Operating Systems Review*, 27(2):10–14, April 1993.

[Pau97] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. Technical Report 440, Computer Laboratory, University of Cambridge, England, December 16, 1997.

[PP96] Algirdas Pakštas and Sonata Pakštienė. Standards: NSK: A Norwegian cryptochip for supersafe communications. *Computer*, 29(2):78–79, February 1996.

[PS00] Adrian Perrig and Dawn Xiaodong Song. A first step on automatic protocol generation. In *Proceedings of Network and Distributed System Security*, February 2000.

[Ros96] A. W. Roscoe. Intensional specifications of security protocols. In *9th IEEE Computer Security Foundation Workshop*, pages 28–38, 1996.

[RSA78] Ronald L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(12):120–126, February 1978.

[Sch96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, second edition, 1996.

[Sel00] Gisbert W. Selke. *Kryptographie: Verfahren, Ziele, Einsatzmöglichkeiten.* O'Reilly Verlag, 2000. (german).

[SH99] Elton Saul and Andrew Hutchison. SPEAR II. In *South African Telecommunications, Networks and Applications Conference (SATNAC'99)*, September 6–8, 1999.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security, CCS'96*, pages 31–37, March 14–15, 1996.

[SW99] Sean W. Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31:831–860, 1999.

[WSB99] Uwe G. Wilhelm, Sebastian Staamann, and Levente Buttyán. Introducing trusted third parties to the mobile agent paradigm. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, LNCS 1603. Springer Verlag, 1999.

[Yee94] Bennet S. Yee. *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University, May 1994.

Index